John von Neumann Institute for Computing

Optimization of an Octree-based 3-D Parallel Meshing Algorithm for the Simulation of Small-Feature Semiconductor Devices

J.J. Pombo, M. Aldegunde, A.J. García-Loureiro

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 439-446, 2006.

http://www.fz-juelich.de/nic-series/volume33

Optimization of an octree-based 3-D parallel meshing algorithm for the simulation of small-feature semiconductor devices

Juan J. Pombo[a], M. Aldegunde[a], A.J. Garcia-Loureiro[a]

[a]Department of Electronics and Computer Science, University of Santiago de Compostela 15782 - Santiago de Compostela (Spain)

1. Abstract

This paper describes an optimized meshing strategy that fits the requirements of a finite element (FEM) simulation software for small-feature semiconductor devices. The main characteristic of these devices is the existence of very small layers of material and transiction regions where several magnitudes of interest can change very abruptly. Because of that, elements in the mesh for the 3D FEM have to be be positioned quite carefully in order to fulfil different size and vertex position requirements. In other hand, as a consecuence of the small details in the devices that should be captured, the number of elements can be moderately high, while geometries, however, are not very complex. In this scene an adapted parallel octree for volume representation, combined with a pattern based algorithm that embeds the tetrahedral mesh in the octree is shown as a feasible solution that performs sucessfully in a parallel computer.

2. Introduction

In the context of the numerical simulation of new semiconductor devices efforts are being made to achieve light-weigthed tools that model correctly the behaviour of small-feature devices. Several kinds of common new transistors are approaching nowadays nanometric features. So is the case of the submicron high electron mobility transistors (HEMT), submicron MOSFETs, bipolar junction transistors (BJT) or heterojunction bipolar transistors (HBT).

One of the difficults that arise in this kind of problems is the very elongated models to consider. The total lenght dimensions in Y axis could be as much as twenty times those on the X axis.

These devices also may present very thin layers of different materials or doping characteristics. In many cases corresponding also to different materials. So nodes of the mesh must be positioned exactly in the boundary.

As a consequence this situation demands special care to guarantee certain quality criteria [9]. The simulation process have to deal with the resolution of equations over very small or stretched regions of materials. In order to guarantee that Finite Element Methods (FEM) based analysis produces adequate results, tetrahedral meshes with special grading control and form factors have to be created. In other case, the meshing process could result in an enormous amount of small elements that causes the performance of the whole simulator to decay drastically.

Modelling the behaviour of these small-geometry devices is very important to develop CAD tools that help to the adjustment of device parameters in the design process [8], [7]. This work describes the efforts to obtain a reliable 3D meshing tool as a part of a complete parallel simulation software in development in our group [6]. The simulation program is able to work well with distributed data, generating distributed tetrahedral meshes suitable for a parallel Finite Element Method (FEM) solver to proceed without important data modifications.

In the present work we focus our attention just in the algorithm for the meshing part of the pro-

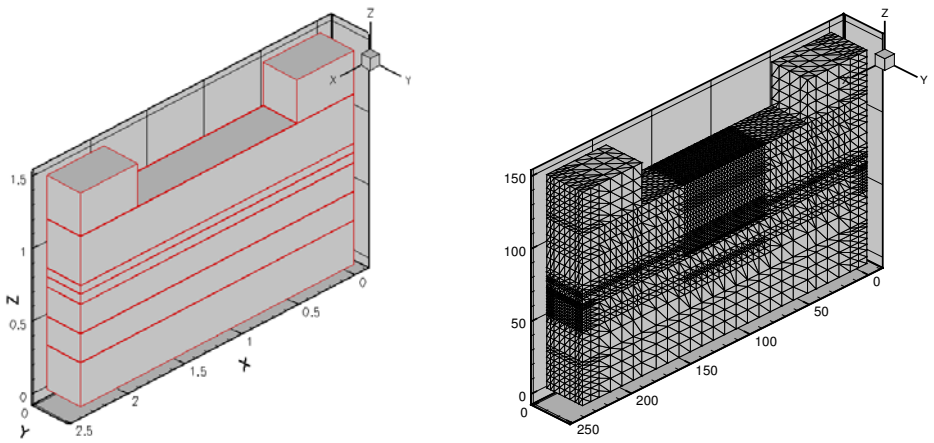
cess. However, the meshes obtained have been tested to ensure that they are suitable for a correct numerical simulation.

The meshing algorithm presented take advantage of the use of a parallel octree based strategy and is able to overcome sucesfully several difficulties related to this kind of problems. It performs adecuately, presents scalability and is suitable for a correct numerical simulation.

3. The semiconductor problem

In order to adapt the resulting mesh to the requirements of the problems at hand some constraints have been considered in the program. In the next paragraphs we summarize and shortly describe some of them.

Multilayer conformity: The problems that we consider in this paper do not present boundaries with very complex geometries and most of the faces are well represented with planar configurations. This is quite common in this kind of simulations and we take advantage of it in several steps. However, new difficulties are caused by the big amount of planar layers existent, some of them extremely close to the others (Figure 1). In case of multilayer structures node occurrences become mandatory in the boundary between different materials. These boundaries are represented as inner faces and have a special consideration across the meshing program.



(a) (b)

Figure 1. (a) Solid model for a HEMT structure and (b) final tetrahedral mesh

Minimized shared mesh: For the distributed tetrahedral mesh, an overlapping must exist between the local meshes assigned to each processor and the “offprocesor” regions (as required by the numerical solver). This fact makes necessary to minimize the amount of elements in the shared region between processor. In other case it would cause degradation of the scalability and the whole performance of the solution process. See Section 5.1 for more details.

Refinement representation: Depending on the problem, different models for representing the level of refinement in the mesh are necessary. Most common refinement criteria are achieved by forcing a constrained size of element along faces or general planes in the 3D structure. However some

more complex criteria are necessary in those cases where general functions describing doping are introduced. The octree strategy requires a special treatment to incorporate these requirements at the octree level.

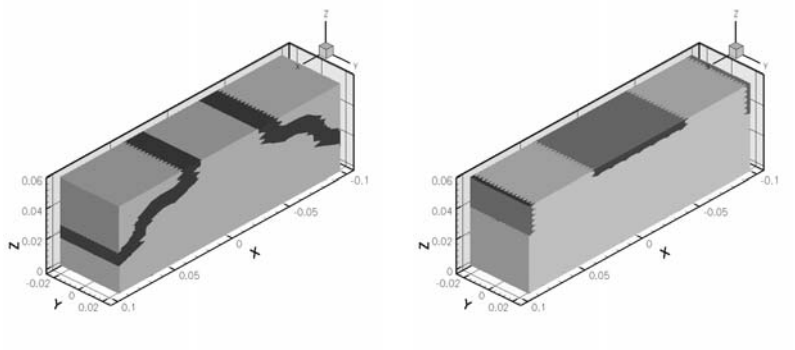


Figure 2. Information of refinement for a MOS transistor. Contacts and other regions of maximum refinement are marked.

Limited mesh size. A compromise exists between good convergence of the FEM problem over the mesh and a reasonable number of elements. It has no sense just increase elements to solve with better accuracy. In 3-D simulators the computations increase enourmously with the number of nodes in the mesh. A correct graded control of the refinement introduced in the last paragraph is a key for preserving good overall performance. This objective is not trivial in an octree-based approach. The thecnique presented here is based in a "default" level of refinement combined with control points.

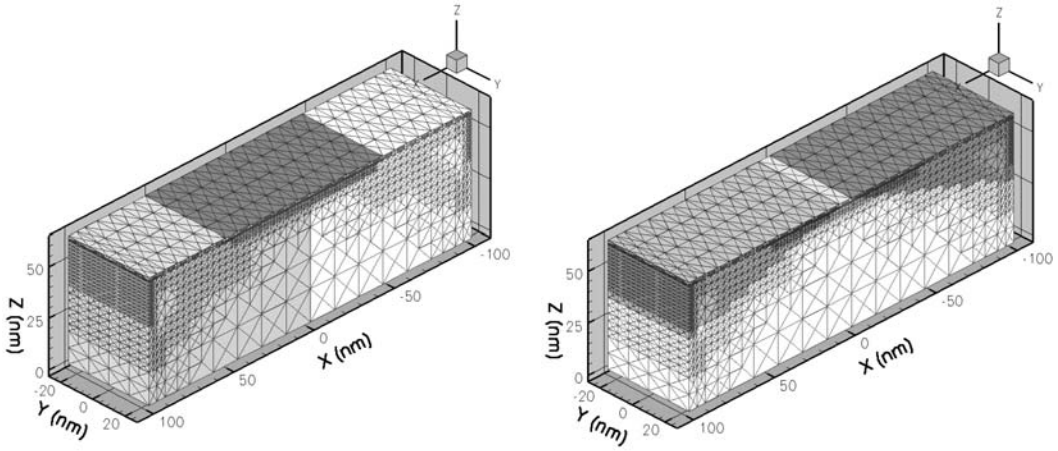
Scaling capability. Very often the elongated sizes in some dimensions of the small-feature transistors make necessary some support for scaling the models previously to the meshing process. This process is desirable for the good performance of the octree based mesher, since it avoids umbalanced trees because of the numerous tree branches outside the solid model. However, the effects of these approaches on the form factor of the tetrahedra establish a limit to this technique when convergence criteria are taken into account. Constraints based on form factors have been introduced.

Automatic load balancing. Parallel generation of the octree is intended to guarantee an adequate load balance of the computing work among processors while minimizing global communications that could degrade algorithm performance. The distribution resulting from this automatic process will be compared to the case of an optimization using mesh partitioning tools like [5].

4. Parallel algorithm

4.1. The octree-method

Octree methods are based on hierarchical space decom position [1]. The process starts constructing an initial cuboid enclosing the whole region. This octant is recursively subdivided into eight parts. The subdivision continues until some refinement criteria are achieved. Only those octants intersecting the boundary of the region or totally enclosed by it are taken into account for further subdivision. Refinement criteria are introduced during the aforementioned process as an input containing a global minimum level of the octree and local specifications. Those criteria are introduced by mean of control points indicating the required octree level in each region (see Figure 3). The goal of reaching these levels guides the creation of the octree.



(a) (b)

Figure 3. Two different mesh distributions for 3 processors in a MOS problem

The use of this data structure allows an easy way to manage the storage of the information needed during the meshing process. Operations which use this information include neighbour finding and boundary adaptation. Both are fundamental parts in the generation of the octree to be meshed. Specially important is the first one, which will be accessed thousands of times (or even millions in case of large meshes) during mesh generation. A description of the neighbour finding algorithm implemented can be found in [4].

4.2. Architecture of the parallel program

We can obtain several benefits from the use of the octree as a model representation. First, the geometric model at hand is subdivided and partitioned in a natural way. The tree is generated in parallel. We take advantage of this situation to divide the load of work among the processors without overheads. Octants of level 0 (root), 1, and 2 are created equal in all the processors. The 64 branches of level 2 in the *octree* are distributed in blocks (Figure 3). In each processor exist only the branches of the octree assigned to itself. In order to access other branches considered "off-processor" in a traversal operation, there exist links in certain octants called *gateways* that allow a fast tracking down of the tree.

The data structure that results, the octree, ensures also a hierarchical and easy to use storage of the information needed during the meshing process. Searching operations across large data arrays are avoided when possible. On the other hand, several algorithms are implemented taking into account the special properties of the octree. We have to be specially careful with the selected data distribution, mainly when we work with a parallel approach to the searching operation. The main kernel that has to be implemented over the octree is the vicinity finding operation. Efforts have been made to code these procedures in an efficient way, since a lot of vicinity tests, about shared faces or vertices between entities, have to be completed during the process. And these tests involve, sometimes, information associated to other processors.

Figure 4 depicts this point. Starting in the most refined leave nodes of the tree somekind of advancing front is created growing in the local octree based on vicinity operations. Visited leave nodes that are not sufficiently refined are partitioned, and neighbourg octants not local to the processor are annotated and sent to the processors that can solve them. This is a one way communication that

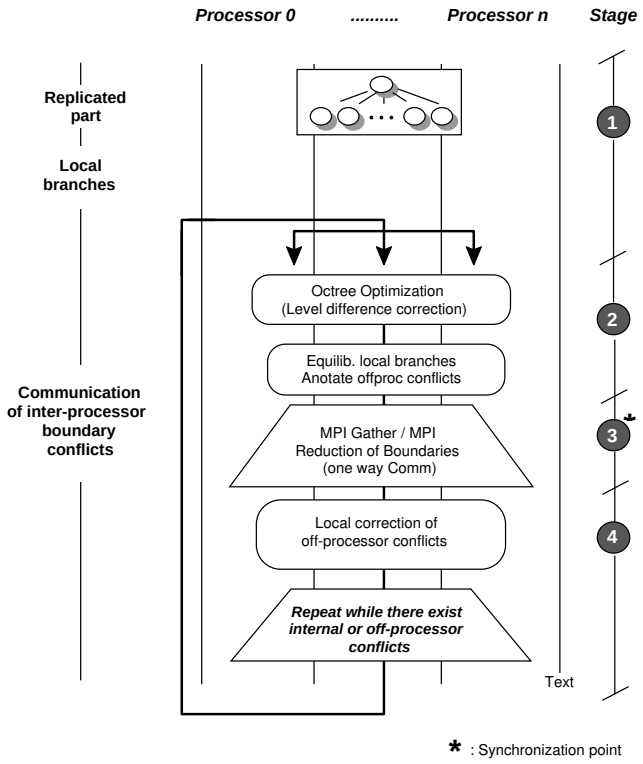


Figure 4. *Octree* construction scheme

do not require response. Each processor then solves their the problem locally (steps 2, 3 and 4 in Figure 4.

5. Tetrahedral mesh generation

Once we have the octree adapted to the geometry, layer constraints and refinement criteria an algorithm to embed the tetrahedral mesh based on the templates method is executed. It is possible to use several algorithms for generating the tetrahedra. The algorithm we select depends on the requirements of the situation. Sometimes we want a faster execution with similar resolution (given by the octree) and sometimes we need a mesh with elements of higher quality. These different methods available are based on the same underlying idea. We generate a basic set of nodes for each terminal octant, and then connect them using a well stablished pattern (Figure 5). We consider in this paper a case where eight vertex are generated for a normal octant. New extra vertices (and elements) could appear in the centers of octant, face or edge, taking into consideration more refined neighbours, and assuring conformality of the mesh.

This strategy, applied to semiconductor devices, reduces the number of nodes and elements in mesh. The only problem for parallelism is that these “extra” nodes are always taken from the most refined neighbours, and sometimes these octants belong to other processor. It seems not feasible just to communicate between processors to obtain the needed coords of the shared node each time the problem appears during the local octree traversal. If we did that the flux of the program would be continuously interrupted. Instead of that, our algorithm creates a structure that solves the eighteen posible directions of vicinity, and is prepared in advance. Creating this structure implies communication with other processors, finding of remote neighbourgh octants, and return of auxiliar coordinates.

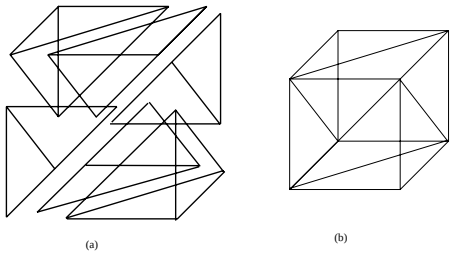


Figure 5. Tetrahedral pattern.

These stages are noted as 2,3 and 4 in Figure(6). It also involves a slight memory overhead. It should be noted that, even though this stages are not the most time consuming steps, they are very costly in terms of scalability. In Section 6 this effect is studied in detail.

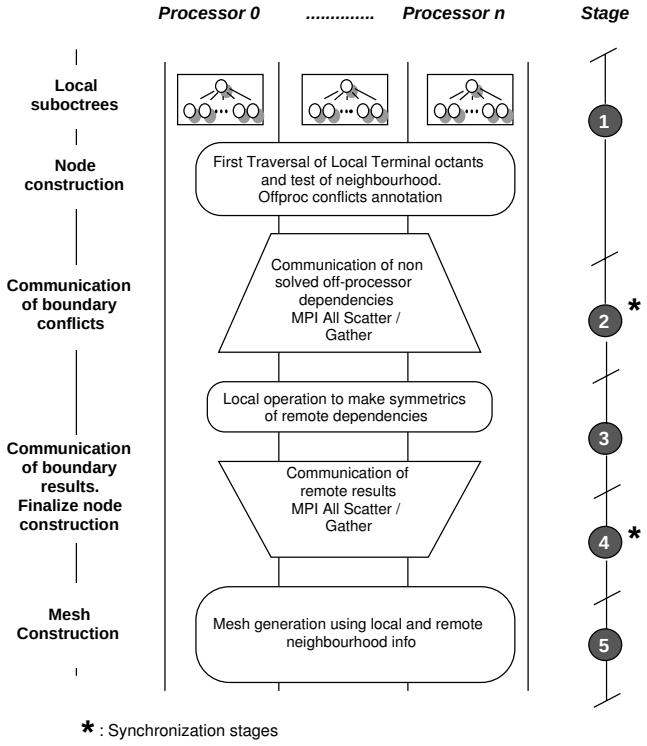


Figure 6. Tetrahedralization scheme

5.1. Analisis of the communication process

Communication between processors occurs in this implementation in several stages along the complete execution. Tuning these stages is very important to get good performance. In any of these steps, when executed with a low number of processors, it is very feasible that communications “all-to-all” could be necesary, since most of the processors can contain neighbour branches of the *octree*. Fortunally, as the number of processors increases, this behavior reachs a limit allowing reasonable scalability.

Since the aforementioned communication stages introduce some kind of syncronization, an ade-

quate load balance is very important. The difficulties to achieve this balance come from the fact that the exact number of *terminal* octants (leave nodes of the *octree*) is only known after obtaining the complete tree, since the generation of the *octree* is completed in a distributed mode. So, the offspring of distributable octant has to be estimated at the beginning of the process, using the refinement control points. When using the mesher in an adaptive scheme [3], this problem is not so important, since changes in the *octree* distribution could be done at the beginning of each new iteraction.

6. Results and Performance

The code has been developed using MPI [2]. It has been executed on a Silicon Graphics Origin 200 and a cluster HP Integrity Superdome. The results are shown in Table 1 and Table 2. A load balance resume is also presented in Table 3.

The labels in the columns of the tables of execution times must be read as follows: OCTGEN, octree generation time. ADJ, time used to adjust octants to boundaries and layers. OPT, process of optimizing the level difference between branches of the *octree* (see Figure 4). TOTAL OCT, total octree generation time. COM(2WAYS), time used for solving neighbourhood offprocessor, there are two steps, send problems, and return of results, (see Figure 6). LOCSIMET, process in the middle of the previous ones, used to compute the problems of offprocessor neighbourhood that can be managed locally. TMESH, complete time of the element generation. TOT T, time used by the whole process.

We can obtain several conclusions checking these results. First of all, it is noticeable that the most consuming part of the algorithm is the preparation of the *octree* structure. The good point is that this part scales very nicely. Once the octants are adjusted in such a manner that neighbour octants differ at most in one level of refinement, defining the tetrahedral elements and nodes involved is a straightforward procedure. This part presents lower scalability but it supposes a small percentage of the complete meshing time.

Load balancing is good enough for not having negative influences on performance.

Scalability is good for small number of processors, but freezes in certain point. This is not such a negative behaviour taking into account the very irregular code that we are analyzing. The key point is that, given a dimension of a problem, we have a limit in the number of processors in which we can obtain important speedups. If the size of the problem increases it is likely that scalability reappears.

We have shown in the tables a quite small problem, in terms of number of terminal octants. However, it makes sense to consider it, since in a FEM simulation based on an adaptive loop the meshing process can be quite time consuming if the number of interactions increase, and mainly when this meshing process is sequential, and data of the mesh has to be partitioned and redistributed in each loop.

In the superdome the effect of the speedy CPU's decreases the scalability. Values of speedups greater than the *np* number appear in the case of the Origin machine. This is not an effect of superscalability, but a cause of the different behaviour in the algorithm when it is executed in parallel, since it is intrinsically parallel.

References

[1] Mark A. Yerry and Mark S. Shephard. Automatic Three-dimensional mesh generation by the Modified-Octree Technique. Int. J. for Num. Methods in Eng. Vol. 20, pp.1965-1990, 1984.

[2] Message Passing Interface Forum. MPI: A Message Interface Standard. Computer Science Department, University of Tennessee. CS-94-230, Knoxville, TN, 1994.

[3] M.S. Shephard, J.E. Flaherty, C.L. Bottasso, H.L. de'Cougny, C. Ozturan, M.L. Simone. Parallel auto-

Execution Times in Origin 200(secs)									
NProc	Execution Time (secs)								**Speedup**
	OCTGEN	ADJ	OPT	Total OCT	COM(2WAYS)	LOCSIMET	TMESH	TOT T	
1	5.05	1.64	0.18	7.9			1.51	9.51	
2	2.67	0.79	0.09	3.55	0.05	0.11	0.16	3.71	2.56
3	2.04	0.62	0.08	2.74	0.34	0.02	0.36	3.10	3.07
4	1.41	0.43	0.12	1.96	0.28	0.03	0.31	2.2	4.32

Table 1
Execution Times for the HEMT transistor in the Origin 200

Execution Times in HP Superdome									
NProc	Execution Time (secs)								**Speedup**
	OCTGEN	ADJ	OPT	Total OCT	COM(2WAYS)	LOCSIMET	TMESH	TOT T	
1	1.02	0.33	0.02	1.37			0.23	1.6	
2	0.56	0.16	0.05	0.77	0.01	0.03	0.12	0.89	1.79
3	0.32	0.10	0.12	0.54	0.06	0.02	0.08	0.62	2.58
4	0.24	0.08	0.08	0.4	0.04	0.02	0.06	0.46	3.58
5	0.20	0.06	0.07	0.33	0.04	0.03	0.07	0.40	4.0
6	0.20	0.06	0.08	0.34	0.05	0.02	0.07	0.40	4.0

Table 2
Execution Times for the HEMT transistor in Cluster HP Integrity Superdome

Load Balance for the HEMT mesh	
Number of processors	**Number of Terminal octants**
1	21384
2	10748/10636
3	6342/5470/5318
4	5318/5374/5238/5264
5	2240/4278/4326/5222/5086
6	2088/2088/4406/4174/4430/3966

Table 3
Number of terminal octants meshing the HEMT problem

matic adaptive analysis. Parallel automatic adaptive analysis. Parallel Computing, Vol. 23, pp.1327-1347, 1996.

[4] J. J. Pombo, T. F. Pena and J. C. Cabaleiro. Parallel Complete Remeshing for Adaptive Schemes. Proc. HPSECA-ICPP, 2001.

[5] G. Karypis and V. Kumar. METIS: A software Package. The University of Minnesota. 1997.

[6] A.J.Garcia-Loureiro, K. Kalna and A. Asenov. 3D Parallel Simulations of Fluctuation Effects in pHEMTs. J. Comp. Elec, Vol.2, pp.369-373, 2003

[7] QMG 1.1 Reference Manual. Computer Science Departament, Cornell University, 1996.

[8] B. H. V. Topping, J. Muylle, P. Ivany, R. Putanowicz and B. Cheng. Finite Element Mesh Generatio, Saxe-Coburg Publications. Kippen, Stitrling, Scotland, 2004.

[9] N. Hitschfeld and M. C. Rivara. Improving the quality of meshes for the simulation of semiconductor devices using Lepp-based algorithms. Int. Journ. for Num. Methods in Enginnering. Vol.5, pp.333-347, 2003.